Slick-dnn Documentation

Kasper Sapala

Jul 01, 2019

Table of Contents

1 Autograd **1**
- 1.1 How it works . . . 1
- 1.2 Fundamental classes . . . 1
- 1.3 Mathematical . . . 2
- 1.4 Activation Functions . . . 4

2 Optimizers **7**
- 2.1 Available Optimizers . . . 7

Python Module Index **9**

Index **11**

CHAPTER 1

Autograd

1.1 How it works

When you write:

```
import numpy as np
from slick_dnn.variable import Variable

a = Variable(np.ones(3))
b = Variable(np.ones(3))

c = a + b
```

New Variable c is created. It's `c.data` is numpy array `[2, 2, 2]`. But it also tracks history of creation.

So it's `backward_function` was set to *`Add.backward`* and it's `backward_variables` was set to `[a, b]`

1.2 Fundamental classes

class slick_dnn.autograd.**Autograd**

Autograd is a base class for all operations made on Variables.

__call__(**variables_list*)

For convenience. One can use all Autograd objects by simply calling them instead of using *`apply`* method.

Parameters **variables_list** – same as in *`forward`* method

Returns what *`apply`* returns

apply(**variables_list*)

Actual creation of new Variable. It calls overwritten *`forward`* method, creates new *`Context`* (same context is used in forward an backward pass) and sets `backward_function` and `backward_variables` for the new Variable

Parameters **variables_list** – any variables, backward function will have to calculate gradients w.r.t all input variables

Returns one variable, the one with tracked history and calculated data

backward(*ctx*, *grad*)
Backward pass. Each Autograd Object must implement it.

Parameters

- **ctx** (Context) – Same context as in forward pass
- **grad** – gradient

Returns gradient w.r.t all inputs

forward(*ctx*, **tensors_list*)
Forward pass of variable. Each Autograd object must implement it.

Parameters

- **ctx** (Context) – Context, classes can save information in them
- **tensors_list** – any list of input tensors

Returns one new tensor

class slick_dnn.autograd.**Context**
This class is for storing information for back propagation. Autograd uses this class instead of using self to allow constructions:

```
relu = ReLU()

b = relu(a)
c = relu(b)
```

That means, that you can use one instance of Autograd class to all of your operations. Without it, the above example would be:

```
relu1 = ReLU()
relu2 = ReLU()

b = relu1(a)
c = relu2(b)
```

save_for_back(**data*)
Saves given data for back propagation.

Parameters **data** (*Any*) – Iterable of any data to save.

1.3 Mathematical

All mathematical operations available for Variables

class slick_dnn.autograd.mathematical.**Add**
Adds given tensors

backward(*ctx*, *grad*)
Backward pass. Each Autograd Object must implement it.

Parameters

- **ctx** (Context) – Same context as in forward pass
- **grad** – gradient

Returns gradient w.r.t all inputs

forward(*ctx*, *tensor1*, *tensor2*)
Forward pass of variable. Each Autograd object must implement it.

Parameters

- **ctx** (Context) – Context, classes can save information in them
- **tensors_list** – any list of input tensors

Returns one new tensor

class slick_dnn.autograd.mathematical.**MatMul**
Matrix multiplication: tensor1 @ tensor2

backward(*ctx*, *grad: numpy.array*)
Backward pass. Each Autograd Object must implement it.

Parameters

- **ctx** (Context) – Same context as in forward pass
- **grad** – gradient

Returns gradient w.r.t all inputs

forward(*ctx*, *tensor1*, *tensor2*)
Forward pass of variable. Each Autograd object must implement it.

Parameters

- **ctx** (Context) – Context, classes can save information in them
- **tensors_list** – any list of input tensors

Returns one new tensor

class slick_dnn.autograd.mathematical.**Mul**
Element-wise multiplication

backward(*ctx*, *grad: numpy.array*)
Backward pass. Each Autograd Object must implement it.

Parameters

- **ctx** (Context) – Same context as in forward pass
- **grad** – gradient

Returns gradient w.r.t all inputs

forward(*ctx*, *tensor1*, *tensor2*)
Forward pass of variable. Each Autograd object must implement it.

Parameters

- **ctx** (Context) – Context, classes can save information in them
- **tensors_list** – any list of input tensors

Returns one new tensor

class slick_dnn.autograd.mathematical.**Sub**
Subtracts given tensors: tensor1-tensor2

backward(*ctx*, *grad*)
Backward pass. Each Autograd Object must implement it.

Parameters

- **ctx** (Context) – Same context as in forward pass
- **grad** – gradient

Returns gradient w.r.t all inputs

forward(*ctx*, *tensor1*, *tensor2*)
Forward pass of variable. Each Autograd object must implement it.

Parameters

- **ctx** (Context) – Context, classes can save information in them
- **tensors_list** – any list of input tensors

Returns one new tensor

1.4 Activation Functions

All activation functions available.

class slick_dnn.autograd.activations.**ArcTan**
Applies the arctan function element-wise.

backward(*ctx*, *grad*)
ArcTan(x)' = $\frac{1}{x^2+1}$

forward(*ctx*, *x*)
ArcTan(x) = $tan^{-1}(x)$

class slick_dnn.autograd.activations.**ReLU**
Applies the ReLU function element-wise.

backward(*ctx*, *grad*)
ReLU(x)' = $\begin{cases} 0 & \text{if } x < 01 \\ \text{if } x > 0 \end{cases}$

forward(*ctx*, *x*)
ReLU(x) = $max(0, x)$

class slick_dnn.autograd.activations.**Sigmoid**
Applies the Sigmoid function element-wise.

backward(*ctx*, *grad*)
Sigmoid(x)' = $\frac{e^{-x}}{(1+e^{-x})^2}$

forward(*ctx*, *x*)
Sigmoid(x) = $\frac{1}{1+e^{-x}}$

class slick_dnn.autograd.activations.**Softmax**
Applies the Softmax function element-wise.

backward(*ctx*, *grad*)
$Softmax(x_i)' = \frac{exp(x_i) * \sum_{j \neq i} exp(x_j)}{(\sum_j exp(x_j))^2}$

forward(*ctx*, *x*) → numpy.array
$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$

class slick_dnn.autograd.activations.**Softplus**

Applies the softplus function element-wise.

backward(*ctx*, *grad*)

Softplus'(x) = $\frac{1}{1+e^{-x}}$

forward(*ctx*, *x*) $\rightarrow$ numpy.array

Softplus(x) = $ln(1 + e^x)$

class slick_dnn.autograd.activations.**Softsign**

Applies the softsign function element-wise.

backward(*ctx*, *grad*)

Softsign'(x) = $\frac{1}{(1+|x|)^2}$

forward(*ctx*, *x*)

Softsign(x) = $\frac{1}{1+|x|}$

class slick_dnn.autograd.activations.**Tanh**

Applies the tanh function element-wise.

backward(*ctx*, *grad*)

Tanh(x)' = $1 - Tanh^2(x)$

forward(*ctx*, *x*)

Tanh(x)

CHAPTER 2

Optimizers

2.1 Available Optimizers

Python Module Index

S

slick_dnn.autograd, 1
slick_dnn.autograd.activations, 4
slick_dnn.autograd.mathematical, 2

Index

Symbols

__call__() (*slick_dnn.autograd.Autograd method*), 1

A

Add (*class in slick_dnn.autograd.mathematical*), 2
apply() (*slick_dnn.autograd.Autograd method*), 1
ArcTan (*class in slick_dnn.autograd.activations*), 4
Autograd (*class in slick_dnn.autograd*), 1

B

backward() (*slick_dnn.autograd.activations.ArcTan method*), 4
backward() (*slick_dnn.autograd.activations.ReLU method*), 4
backward() (*slick_dnn.autograd.activations.Sigmoid method*), 4
backward() (*slick_dnn.autograd.activations.Softmax method*), 4
backward() (*slick_dnn.autograd.activations.Softplus method*), 5
backward() (*slick_dnn.autograd.activations.Softsign method*), 5
backward() (*slick_dnn.autograd.activations.Tanh method*), 5
backward() (*slick_dnn.autograd.Autograd method*), 2
backward() (*slick_dnn.autograd.mathematical.Add method*), 2
backward() (*slick_dnn.autograd.mathematical.MatMul method*), 3
backward() (*slick_dnn.autograd.mathematical.Mul method*), 3
backward() (*slick_dnn.autograd.mathematical.Sub method*), 3

C

Context (*class in slick_dnn.autograd*), 2

F

forward() (*slick_dnn.autograd.activations.ArcTan method*), 4
forward() (*slick_dnn.autograd.activations.ReLU method*), 4
forward() (*slick_dnn.autograd.activations.Sigmoid method*), 4
forward() (*slick_dnn.autograd.activations.Softmax method*), 4
forward() (*slick_dnn.autograd.activations.Softplus method*), 5
forward() (*slick_dnn.autograd.activations.Softsign method*), 5
forward() (*slick_dnn.autograd.activations.Tanh method*), 5
forward() (*slick_dnn.autograd.Autograd method*), 2
forward() (*slick_dnn.autograd.mathematical.Add method*), 3
forward() (*slick_dnn.autograd.mathematical.MatMul method*), 3
forward() (*slick_dnn.autograd.mathematical.Mul method*), 3
forward() (*slick_dnn.autograd.mathematical.Sub method*), 4

M

MatMul (*class in slick_dnn.autograd.mathematical*), 3
Mul (*class in slick_dnn.autograd.mathematical*), 3

R

ReLU (*class in slick_dnn.autograd.activations*), 4

S

save_for_back() (*slick_dnn.autograd.Context method*), 2
Sigmoid (*class in slick_dnn.autograd.activations*), 4
slick_dnn.autograd (*module*), 1
slick_dnn.autograd.activations (*module*), 4
slick_dnn.autograd.mathematical (*module*), 2
Softmax (*class in slick_dnn.autograd.activations*), 4
Softplus (*class in slick_dnn.autograd.activations*), 5

Softsign (*class in slick_dnn.autograd.activations*), 5
Sub (*class in slick_dnn.autograd.mathematical*), 3

T

Tanh (*class in slick_dnn.autograd.activations*), 5